Carrera: **Analista Programador**

PROGRAMACIÓN ORIENTADA A OBJETOS

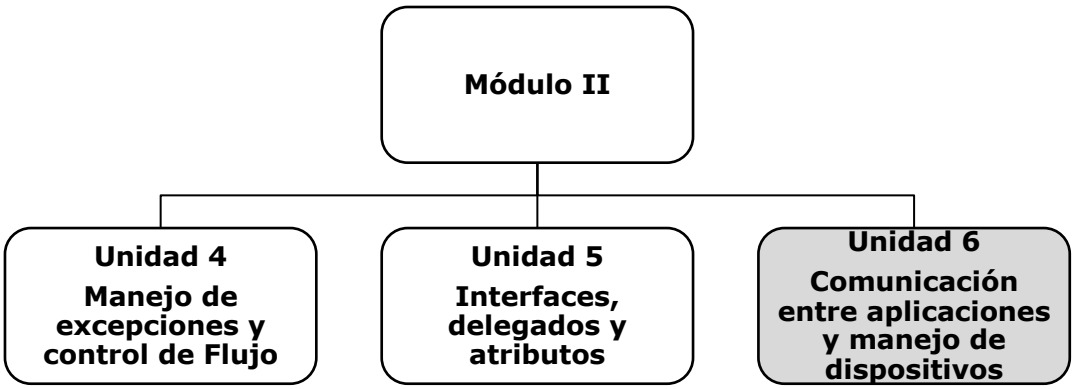
Módulo II

Programación de aplicaciones utilizando la técnica de la Programación Orientada a Objetos

Unidad 6

Comunicación entre aplicaciones y manejo de dispositivos

Docente titular y autor de contenidos: Prof. Ing. Darío Cardacci



Presentación

En esta unidad trabajaremos sobre las formas de desarrollar aplicaciones cliente – servidor y las características generales de los puertos de la computadora. Para las comunicaciones entre aplicaciones se utilizarán sockets. Esperamos que descubrir estas características le abra un abanico posibilidades para potenciar su programación.

Por todo lo expresado hasta aquí es que esperamos que usted, a través del estudio de esta unidad, se oriente hacia el logro de las siguientes metas de aprendizaje:

- Conceptualizar las características de un sistema cliente – servidor.
- Comprender cómo se pueden programar los socket de una computadora.
- Identificar, explicar y valorar las características más distintivas de los puertos de la computadora.

Los siguientes **contenidos** conforman el marco teórico y práctico que contribuirá a alcanzar las metas de aprendizaje propuestas:

Esquema cliente - servidor. Configuración de aplicaciones remotas en una red. Formas de compartir información entre aplicaciones. Pasaje de información batch Vs. On line.

Conceptos básicos de protocolos. TCP. UDP. Concepto sobre IP, TCP/IP. Concepto de software cliente – servidor. Ventajas y desventajas. Distribución de procesos y Almacenamientos. Sistema de mensajería de un sistema cliente – servidor.

Utilización de Sockets. Sockets de clientes sincrónicos y asincrónicos. Socket de servidores sincrónicos y asincrónicos.

Manejo de puertos.

Características del puerto paralelo. Arquitectura física. Estructura lógica.

Características del puerto serial. Arquitectura física. Estructura lógica.

A continuación, le presentamos un detalle de los contenidos y actividades que integran esta unidad. Usted deberá ir avanzando en el estudio y profundización de los diferentes temas, realizando las lecturas requeridas y elaborando las actividades propuestas, algunas de desarrollo individual y otras para resolver en colaboración con otros estudiantes y con su profesor.

Contenidos y Actividades

1. COMUNICACIÓN ENTRE APLICACIONES

Lectura requerida

- http://msdn.microsoft.com/en-s/library/system.net.sockets.socket.aspx

2. COMUNICACIÓN POR SOCKET

Lectura requerida

- http://msdn.microsoft.com/en-s/library/system.net.sockets.socket.aspx

3. PUERTOS DE LA COMPUTADORA

Lectura requerida

- Montefinal, Fabián H.; Cardacci, Darío G. Conceptos básicos sobre electricidad: electrónica y puertos de la PC. 2a.ed.-- Buenos Aires: Universidad Abierta Interamericana, c2006. 78 páginas

Para el estudio de estos contenidos usted deberá consultar la bibliografía que aquí se menciona.

BIBLIOGRAFÍA OBLIGATORIA

- Montefinal, Fabián H.; Cardacci, Darío G. Conceptos básicos sobre electricidad: electrónica y puertos de la PC. 2a.ed.-- Buenos Aires: Universidad Abierta Interamericana, c2006. 78 páginas

Bibliografía Ampliatoria

- Zacker, Craig; Rourke, John. PC Hardware: manual de referencia.-- Madrid : McGraw-Hill Interamericana, c2001. XXIII, 752 páginas, 1 CD-Rom

Links a temas de interés

MSDN. Microsoft Developer Network.

http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.aspx

¿QUÉ SON LOS SOCKET EN PROGRAMACIÓN?

Un socket es un método de comunicación entre un programa cliente y un servidor. Entonces podemos decir que los sockets, son el "tunel" de comunicación entre dos aplicaciones.

Lo/a invitamos ahora a comenzar con el estudio de los contenidos que conforman esta unidad.

1. COMUNICACIÓN ENTRE APLICACIONES

Para que una comunicación se produzca se necesitan tres elementos fundamentales: **un emisor**, **un canal** y **un receptor**.

El emisor es quien envía los datos, por ejemplo una computadora. El canal es el medio por el cual se envían, puede ser alámbrico (cable coaxial), inalámbrico (micro ondas) u óptico (fibra óptica) y el receptor podría ser otra computadora que recibe lo enviado.

4. ¿Qué es un protocolo?

Para que el emisor y el receptor puedan transferirse datos también debe existir un **protocolo** de comunicación. Un protocolo es un conjunto de reglas que las computadoras que se comunican utilizan. En él se estandarizan los mensajes que se enviarán entre ambas para de esta manera poder entender lo que cada una envía. Las reglas que establece un protocolo permiten definir la sintaxis, la semántica y la sincronización de lo que se está transmitiendo.

5. ¿Qué protocolo usa una red de área local?

Ejemplo de protocolos son **TCP, UDP, IP** etc. El protocolo **TCP** es un protocolo utilizado para el control de transmisiones. Trabaja en la capa denominada de transporte. Se puede utilizar TCP para crear conexiones seguras entre computadoras y enviar datos. Este protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. TCP puede distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto de puerto. **UDP** (User Datagram Protocol)

IP ⇒ Espacio de Memoria
Puerto ⇒ Puntero a ese espacio de memoria

TCP ⇒ Con control de errores, es mucho más lento.
UDP ⇒ Sin control de errores, por eso es más rápido

Ejemplo de aplicaciones TCP:
Ejemplo de aplicaciones UPD: Streaming (NetFlix)

6. ¿Qué protocolo usa internet?

también es un protocolo que trabaja a nivel de transporte. Para el intercambio de datos utiliza datagramas. No realiza control de flujo, pues no hay confirmación de entrega o recepción. **IP** (Internet Protocol) es un protocolo no orientado a la conexión que trabaja a nivel de red. Los datos son enviados en bloques conocidos con el nombre de paquetes. Las computadoras se identifican por una dirección IP que es un número que identifica de manera única a una computadora.

7. ¿Qué hace el protocolo IP?

2. ¿Qué significa pasar información batch?

3. ¿Qué significa pasar información on line?

Un valor agregado muy importante en el desarrollo de aplicaciones informáticas es que estas puedan estar interconectadas con el objetivo de que se transfieran información. Existen muchas formas en las que dos aplicaciones pueden hacerlo pero dos bien diferenciadas son las que se conocen como **"modo batch"** y **"modo on line".** El modo batch o por lotes como su nombre lo indica, para realizar la transferencia de información previamente acumula una determinada cantidad de datos (un lote), se gestiona la conexión, luego realiza la transferencia y finalmente se desconecta hasta la transferencia del próximo lote. El modo on line se gestiona la conexión y esta se mantiene, los datos se envían a medida que se producen.

1. ¿Qué características posee un esquema cliente - servidor?

En general cuando desarrollamos software y este se conecta con otro software para obtener datos denominamos cliente a quien solicita los datos y servidor a quien los brinda. En realidad, hoy en día la mayoría de las aplicaciones trabajan como cliente y servidor ya que son capaces de brindar ciertos servicios y datos y en otro momento de solicitar colaboración o información remota.

En particular en este curso no interesa comunicar aplicaciones haciendo uso de los socket, tema que desarrollaremos en el siguiente punto.

2. COMUNICACIÓN POR SOCKET

10. ¿Qué es un socket?

Cómo primer paso definiremos que entendemos por **socket.** Existen varias definiciones pero considerando que para ejemplificar estos conceptos utilizaremos los protocolos TCP e IP, tomaremos como sinónimo de socket a un par **Dirección IP** más un **puerto** de la computadora.
Una dirección **IP** permite identificar una computadora en la red y un puerto es una interfaz lógica para la entrada y salida de información. El puerto se identifica con un número entero. Existen números reservados para determinados servicios o aplicaciones. Por ejemplo el puerto 80 para HTTP.
Por ejemplo la IP 192.168.1.1 y el puerto 5050 conformarían un socket.
Para explicar como funciona nos sustentaremos en un ejemplo.

VER EJEMPLO 38

La idea en este ejemplo es que dos aplicaciones se puedan comunicar y pasarse datos, en principio podría ser cualquier dato, pero comenzaremos emulando un chat punto a punto. Tendremos que construir un programa que

Si tuviera conectados 5 clientes, por el lado del servidor, tendría corriendo 7 procesos: el principal y 6 subprocesos más (1 que resuelve la escucha del puerto más 1 por cada cliente conectado).

opere como cliente y otro que lo haga como servidor, al menos en el momento que se establece la conexión ya que luego, cuando se envía y reciben datos, el rol de cada aplicación es indistinto. Para poder lograrlo es muy importante reconocer tres momentos muy marcados en la generación de estas aplicaciones: **la configuración**, **la conexión** y **el envío/recepción** de datos.

Cabe aclarar que para el ejemplo utilizamos las clases **cliente** y la clase **servidor.**

La clase **cliente** posee:

PROPIEDADES:

Conectado	Flag que indica el estado de la conexión
LocalEndPoint	Si el cliente está conectado permite obtener la IP y el puerto local
RemoteEndPoint	Si el cliente está conectado permite obtener la IP y el puerto del servidor

METODOS:

Conectar	Envía una solicitud de conexión al servidor y, si la misma es exitosa, inicia un thread para escuchar los mensajes provenientes del mismo.
EnviarDatos	Envía un mensaje al servidor

EVENTOS:

ConexionTerminada	Se dispara cuando la conexión se termina.
DatosRecibidos	Se dispara cuando el cliente recibe datos.

FUNCIONES PRIVADAS:

LeerSocket	Rutina que se queda esperando la recepción de datos.

La clase **servidor** posee:

PROPIEDADES:

PuertoDeEscucha	Permite obtener el puerto de escucha del servidor

METODOS:

Escuchar	Coloca al servidor en modo escucha esperando la conexión de un cliente.
Cerrar	Cierra la conexión con todos los clientes conectados.
EnviarDatos	Permite enviar datos a todos los clientes conectados.

EVENTOS:

NuevaConexión	Se dispara cuando se recibe una nueva petición de conexión por parte de un cliente.
ConexionTerminada	Se dispara cuando una conexión con un cliente se termina.
DatosRecibidos	Se dispara cuando el servidor recibe datos provenientes de alguno de los clientes conectados.

FUNCIONES PRIVADAS:

LeerSocket	Rutina que se queda leyendo la recepción de datos por parte de un cliente.

EsperarCliente	Rutina que se queda leyendo un puerto a la espera de que un cliente solicite conectarse

ESTRUCTURAS:

InfoDeUnCliente	Estructura que permite guardar el socket y el thread asociados a un cliente.

Es importante que nuestros proyectos de Cliente y Servidor tengan disponibles los siguientes namespace:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
```

CONFIGURACIÓN DEL CLIENTE Y EL SERVIDOR

La **configuración en el cliente** se logra por medio del siguiente código. Primero instanciamos un cliente y nos suscribimos a sus eventos:

```
cliente = new Cliente();
cliente.ConexionTerminada += Cliente_ConexionTerminada;
cliente.DatosRecibidos += Cliente_DatosRecibidos;
```

Luego obtenemos la IP y el puerto del servidor de las cajas de texto en el formulario de la aplicación. En nuestro caso la IP será 127.0.0.1 y el puerto 8050. Dado que el puerto debe ser un entero y la caja de texto contiene un string debemos primero parsear el texto ingresado por el usuario, y mostrar un mensaje de error si el valor no es un entero válido. Una vez obtenidos los parámetros de conexión correctos los pasamos como argumentos del mensaje Conectar.

```
int puerto;
if (!int.TryParse(txtPuerto.Text, out puerto))
{
      MessageBox.Show("El puerto ingresado no es válido", Text);
      return;
}
string ip = txtIP.Text;
cliente.Conectar(ip, puerto);
```

La **configuración en el servidor** se logra por medio del siguiente código. Primero instanciamos un servidor pasando como parámetro del constructor el puerto 8050 y luego nos suscribimos a los eventos del servidor:

```
servidor = new Servidor(8050);

servidor.NuevaConexion += Servidor_NuevaConexion;
servidor.ConexionTerminada += Servidor_ConexionTerminada;
servidor.DatosRecibidos += Servidor_DatosRecibidos;
```

CONEXIÓN DEL CLIENTE Y EL SERVIDOR

Para lograr esto primero hay que poner al servidor a escuchar por el puerto 8050.

```
servidor.Escuchar();
```

El método Escuchar básicamente lo que hace es crear un objeto **TcpListener** que opere en el puerto de escucha configurado y ponerlo a funcionar con Start, pero además delega lo que se está ejecutando a un procedimiento denominado **EsperarCliente** que se ejecuta en un subproceso diferente. Este subproceso alberga el código que se quedará esperando la conexión de un nuevo cliente como puede observarse a continuación.

```
public void Escuchar()
{
	listener = new TcpListener(IPAddress.Any, PuertoDeEscucha);
	listener.Start();

	listenerThread = new Thread(EsperarCliente);
	listenerThread.IsBackground = true;
	listenerThread.Start();
}
```

```
private void EsperarCliente()
{
	while (true)
	{
		var socket = listener.AcceptSocket();
		var endPoint = socket.RemoteEndPoint as IPEndPoint;

		var thread = new Thread(() => LeerSocket(endPoint));
		thread.IsBackground = true;

		clientes[endPoint] = new InfoDeUnCliente()
		{
			Socket = socket,
			Thread = thread,
		};

		NuevaConexion?.Invoke(this,
					new ServidorEventArgs(endPoint));
		thread.Start();
	}
}
```

En el código del bloque anterior se puede observar que cada vez que un cliente se conecta se genera un nuevo subproceso para recibir los mensajes que este cliente envíe al servidor. Cada subproceso que se utilice para la conexión con un cliente utiliza un puerto distinto, de esta manera el puerto 8050 del servidor siempre queda liberado a la espera de una nueva conexión.

Mientras el servidor queda a la escucha el cliente realiza la petición de conexión.

```
public void Conectar(string ip, int puerto)
{
	if (Conectado) return;

	socket = new Socket(SocketType.Stream, ProtocolType.Tcp);
	socket.Connect(ip, puerto);
	Conectado = true;

	thread = new Thread(LeerSocket);
	thread.IsBackground = true;
	thread.Start();
}
```

De forma análoga al procedimiento del servidor, el cliente inicia un subproceso dedicado a leer del socket los mensajes provenientes del servidor.

ENVÍO Y RECEPCIÓN DE DATOS

Para el envío de datos del Cliente al Servidor este lo hace de la siguiente manera:

```
public void EnviarDatos(string datos)
{
	if (!Conectado) return;

	byte[] bytes = Encoding.UTF8.GetBytes(datos);
	socket?.Send(bytes);
}
```

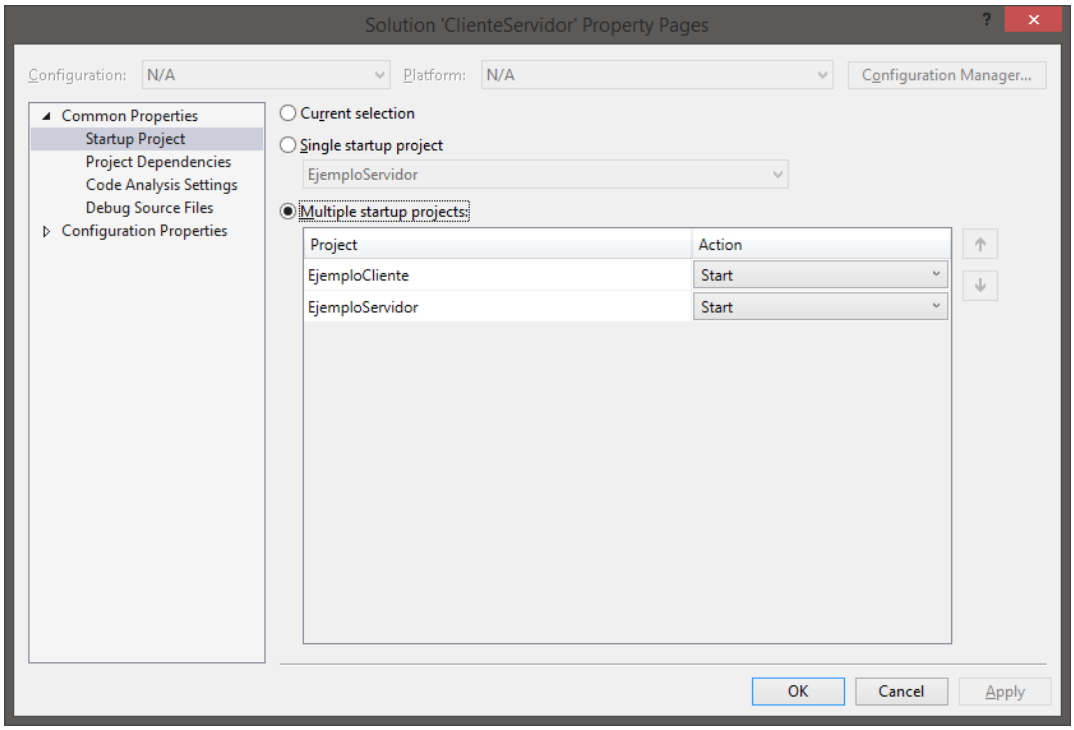
En el bloque de código anterior se puede observar como los datos que se desean enviar se reciben en el parámetro **datos** que es de tipo **string,** luego se transforman a un vector de Bytes y finalmente se envían por medio del método **Send** utilizando la variable **socket**, que es un **Socket** que apunta al servidor.

Por su lado cuando el servidor recibe los datos opera el siguiente código:

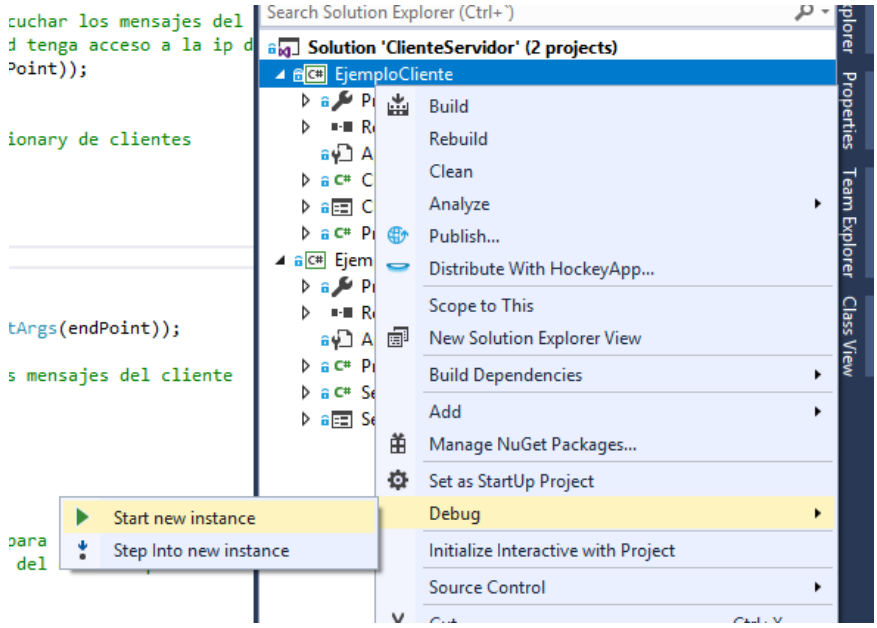
```
private void LeerSocket(IPEndPoint endPoint)
{
	var buffer = new byte[100];
	var cliente = clientes[endPoint];

	while (cliente.Socket.Connected)
	{
		try
		{
			int cantRecibida = cliente.Socket.Receive(buffer, buffer.Length, SocketFlags.None);
			if (cantRecibida > 0)
			{
				var datos = Encoding.UTF8.GetString(buffer, 0, cantRecibida);
				DatosRecibidos?.Invoke(this, new DatosRecibidosEventArgs(endPoint, datos));
			}
			else
			{
				ConexionTerminada?.Invoke(this, new ServidorEventArgs(endPoint));
				break;
			}
		}
		catch
		{
			if (!cliente.Socket.Connected)
			{
				ConexionTerminada?.Invoke(this, new ServidorEventArgs(endPoint));
				break;
			}
		}
	}

	clientes.TryRemove(endPoint, out cliente);
}
```

Le proponemos que ejecute los programas cliente y servidor, configure el cliente en la IP 127.0.0.1 puerto 8050 y pruebe como se pueden intercambiar mensajes. Recomendamos revisar el código del ejemplo ya que el mismo se encuentra comentado en mayor detalle que en el presente documento.

Para poder ejecutar tanto el cliente como el servidor en simultáneo deberá configurar la solución para que ambos proyectos se ejecuten en el inicio.



Si lo desea, también puede ejecutar múltiples instancias del proyecto cliente seleccionando en la opción del menú correspondiente.



3. PUERTOS DE LA COMPUTADORAS

En particular nos interesa analizar de manera introductoria tres puertos de la PC para comprender sus características físicas, lógicas y como se podrían llegar a aprovechar.

Los puerto a analizar son: el **puerto paralelo,** el **puerto serie** y **el puerto USB.**

PUERTO PARALELO

El **puerto paralelo** recibe su nombre por la forma en que los datos son enviados. En general los sistemas paralelos utilizan ocho líneas de datos para transmitir un byte a la vez, y fueron utilizados hace un tiempo fundamentalmente para la conexión de impresoras. No obstante podríamos adaptar este puerto para tener ocho señales y comandar otros dispositivos electrónicos o electromecánicos que nos interesen.

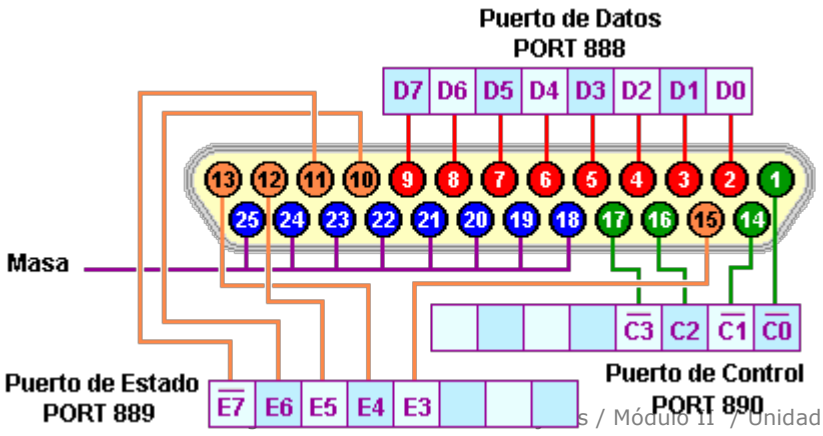
El hecho de mandar ocho bits simultáneamente implica que se deba prestar especial atención a la sincronización de estos bits (deben llegar todos en un mínimo espacio de tiempo). Esto hace que las distancias que se pueden cubrir con este tipo de puerto sean relativamente pequeñas. El hardware requerido para el puerto paralelo está establecido por el estándar IEEE 1284. Este estándar internacional establece tres tipos de conectores para la interfaz:

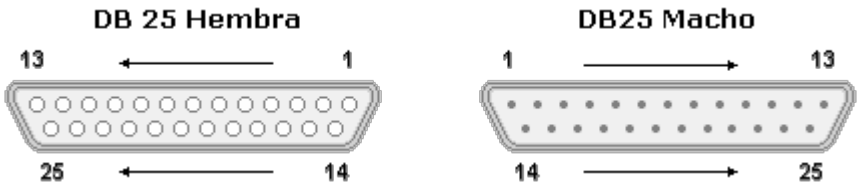
El 1284 tipo A: es un conector hembra de 25 pines de tipo D (DB25-S).

El 1284 tipo B que es un conector de 36 pines de tipo centronics y lo encontramos en la mayoría de las impresoras.

El 1284 tipo C, se trata de un conector similar al 1284 tipo B pero más pequeño, además se dice que tiene mejores propiedades eléctricas y mecánicas y es el recomendado para nuevos diseños.

El conector DB25 es uno de los más difundidos para este tipo de puertos, su aspecto es el siguiente:





PUERTO SERIE

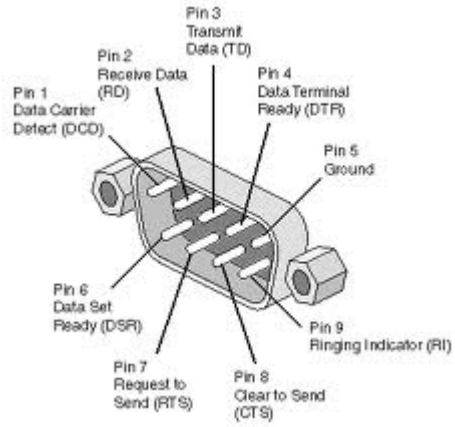
El **puerto serie** de comunicaciones es utilizado para conectar equipamientos que esperan recibir la información bit a bit. El puerto serie de nuestra computadora posee una iterfaz física con un conector. Existen varios estándares para la comunicación serie por ejemplo RS-232, FireWire, Serial ATA y USB en todas sus versiones.

En general es muy importante la manera en que los puertos serie protocolizan la información que envían y reciben debido a que como lo hacen bit a bit los dispositivos deben saber cuando comienza y termina una trama de datos.

Existen varias formas de comunicación en serie: simplex, duplex, full duplex.

En el modo simplex la comunicación es unidireccional existiendo un emisor y un receptor. En el modo duplex ambos equipos pueden funcionar como emisor o receptor pero la información no puede fluir en ambos sentidos al mismo tiempo. Finalmente en el modo full duplex es igual al anterior pero la información puede fluir en ambos sentidos.

Fichas DB-9 para conexiones seriales.



Lectura requerida

.

Guía para la lectura

1. ¿Qué características posee un *esquema cliente - servidor*?
2. ¿Qué significa pasar información *batch*?
3. ¿Qué significa pasar información on line?
4. ¿Qué es un *protocolo*?
5. ¿Qué protocolo usa una red de área local?
6. ¿Qué protocolo usa internet?
7. ¿Qué hace el *protocolo IP*?
8. ¿Qué ventajas tiene distribuir procesos?
9. ¿Qué ventajas tiene distribuir almacenamientos?

8. Los subprocesos se pueden usar para dividir una tarea en unidades más pequeñas que se pueden ejecutar simultáneamente.

10. ¿Qué es un *socket*?
11. ¿Qué característica posee un *socket sincrónico*?
12. ¿Qué característica posee un *socket asincrónico*?
13. ¿Qué objeto se puede utilizar para construir un navegador?
14. ¿Para qué se utilizan los *puertos* de la pc?
15. ¿Qué puertos posee una PC?
16. ¿Cómo funciona el *puerto paralelo*?
17. ¿Cómo funciona el *puerto serie*?
18. ¿Cómo funciona el *puerto USB*?
19. ¿Qué es la *domótica*?

1. Tomando como base el ejercicio de cliente servidor desarrollado como ejemplo adaptarlo para que:
    a. Los mensajes enviados entre el cliente y el servidor se vean en cajas de texto multilineales como si fuera un Chat
    b. Que un cliente le pueda enviar un mensaje a todos los clientes conectados o a un cliente en particular.
    c. Que el servidor le pueda enviar un mensaje a todos los clientes o a uno en particular

EVALUACIÓN PARCIAL

Propuesta para la Integración del MóduloII

Ha llegado el momento de la evaluación parcial del Módulo.

Encontrará el documento con las consignas para su realización en el link correspondiente del campus virtual.

Consulte el Cronograma de la Asignatura para ajustar su producción a los tiempos previstos por el profesor tutor.

Recuerde que esta instancia es obligatoria y, como tal, su realización constituye un requisito para la presentación al examen final. Encontrará más detalles sobre la misma en la Unidad Introductoria de la Asignatura.

Cierre de la unidad

Un brindis virtual?